UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS)

Tesis doctoral

BIG DATA MEETS HIGH PERFORMANCE COMPUTING: GENOMICS AND NATURAL LANGUAGE PROCESSING AS CASE STUDIES

Presentada por:

José Manuel Abuín Mosquera

Dirigida por:

Juan Carlos Pichel Campos
Tomás Fernández Pena

Santiago de Compostela, septiembre de 2017

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

Juan Carlos Pichel Campos, Profesor Contratado Doctor del Área de Arquitectura de Computadores de la Universidad de Santiago de Compostela

Tomás Fernández Pena, Profesor Titular del Área de Arquitectura de Computadores de la Universidad de Santiago de Compostela

HACEN CONSTAR:

Que la memoria titulada **Big Data meets High Performance Computing: Genomics and Natural Language Processing as case studies** ha sido realizada por **José Manuel Abuín Mosquera** bajo nuestra dirección en el Centro Singular de Investigación en Tecnoloxías da Información de la Universidade de Santiago de Compostela, y constituye la Tesis que presenta para optar al título de Doctor.

Santiago de Compostela, septiembre de 2017

Juan Carlos Pichel Campos
Director/a de la tesis

Tomás Fernández Pena
Director/a de la tesis

José Manuel Abuín Mosquera
Autor/a de la tesis

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

Juan Carlos Pichel Campos, Profesor Contratado Doctor del Área de Arquitectura de Computadores de la Universidad de Santiago de Compostela

Tomás Fernández Pena, Profesor Titular del Área de Arquitectura de Computadores de la Universidad de Santiago de Compostela

como Director/res de la tesis titulada:

Big Data meets High Performance Computing: Genomics and Natural Language Processing as case studies

Por la presente DECLARAN:

Que la tesis presentada por Don **José Manuel Abuín Mosquera** es idónea para ser presentada, de acuerdo con el artículo 41 del *Regulamento de Estudos de Doutoramento*, por la modalidad de compendio de ARTÍCULOS, en los que el doctorando ha tenido participación en el peso de la investigación y su contribución fue decisiva para llevar a cabo este trabajo. Y que está en conocimiento de los coautores, tanto doctores como no doctores, participantes en los artículos, que ninguno de los trabajos reunidos en esta tesis serán presentados por ninguno de ellos en otras tesis de Doctorado, lo que firmo bajo mi responsabilidad.

Santiago de Compostela, septiembre de 2017

Juan Carlos Pichel Campos
Director/a de la tesis

Tomás Fernández Pena
Director/a de la tesis

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

A Martín e a Verónica

Por cambiarme a vida. De boa, a mellor

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

The two most important days in your life are the day you are born and the day you find out why.

Mark Twain

There are things known and there are things unknown, and in between are The Doors

Jim Morrison

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

Agradecementos

Non podo comezar esta sección doutra maneira que non sexa agradecendo a Pichel e a Tomás por darme esta oportunidade. Primeiro co proxecto de PLN e despois coa bolsa FPI. Sen o seu apoio dende un principio esta tese non sería posible. Ademais diso nunca deixaron de recibirme amablemente sempre que petaba na súa porta cun novo problema (ou non tan novo).

Quero tamén agradecer ó resto do Grupo de Arquitectura de Computadores da USC, en especial a Fran, Caba e Dora. Ós primeiros porque a miña andanza no mundo "paralelo" empezou con eles durante o meu TFM, e a Dora porque dela aprendín moitísimo no referente a docencia e á vida universitaria como investigador. Así mesmo, tamén darlle as gracias a Pablo Gamallo e a Jorge Amigo, pola súa axuda e sempre boa disposición coas miñas preguntas no tocante ás súas especialidades, PLN e Bioinformática respectivamente.

Non podo esquecerme do resto de compañeiros na miña vida profesional. Comezando, primeiro, polo Cesga, onde me deron a miña primeira oportunidade. Alí tiven a sorte de coñecer e aprender de grandes profesionais e amigos. María José, Manuel Gromaz, Silvia, Cecilia e Javi. Despois, co salto ó Departamento de Matemática Aplicada da USC, tiven o luxo de traballar con Alfredo Bermúdez, Julio, Chuco, Ángel e Víctor, dos cales aprendín tanto tecnicamente como coma persoa, e cos cales gardo grandes recordos.

E no tocante ó CiTIUS, pouco que dicir, e ó mesmo tempo, moito. Esta tese non tería visto o fin sen as longas conversas, apoios, comidas e cafés compartidos con Esteban, Bea, Julio, Diego Rodríguez, Guillermo, Vanesa, Rodrigo e Carlos Bran.

I would also like to thank the High Performance Computing Research Group at the Johannes Gutenberg Univertisät for the great experience in the three and a half months I spent there. I learned a lot from them, specially from Bertil Schmidt, Tu Tuan Tran, Christian Hundt, André Müller and André Weißenberger.

Tamén quero darlle as gracias, como non, á miña familia, por estar sempre presente durante o desenvolvemento deste traballo. Ademais, quero agradecer ás comunidades de StackOverflow, Wikipedia e GitHub, así coma a Linus Torvalds por Linux e a Dennis Ritchie pola linguaxe C e por Unix.

Para rematar, quero agradecer á Xunta de Galicia pola súa financiación no proxecto "Computación de Altas Prestacións para o Procesamento da Linguaxe Natural" (EM2013/041), coa cal estiven contratado, e ó Ministerio de Economía y Competitividad pola financiación do proxecto "Soluciones Hardware y Software para la Computación de Altas Prestaciones" (TIN2013-41129-P) e a correspondente beca FPI (BES-2014-067914). Tamén, como non, agradecer ó Cesga e a Amazon AWS pola posibilidade de empregar as súas instalacións.

Santiago de Compostela, septiembre de 2017

Resumen

En los últimos años las tecnologías Big Data han tenido un auge enorme dentro del mundo de la industria y la investigación. Esto se ha debido, principalmente, a su capacidad para realizar el procesamiento de grandes volúmenes de datos usando arquitecturas paralelas de un modo sencillo, eficiente y completamente transparente para el usuario. Dicho de otro modo, las tecnologías Big Data han acercado la programación paralela clásica en memoria distribuida a un público más general, facilitando en gran medida su adopción sin una pérdida importante de rendimiento, y ese es parte de su éxito.

Dicho éxito se manifiesta en su uso en centros de investigación punteros a nivel mundial. Por ejemplo en el CERN, se usan tecnologías Big Data para procesar los datos producidos por el Gran Colisionador de Hadrones (LHC), o la NASA, que utiliza este tipo de tecnologías para tratar datos recibidos por grandes telescopios instalados en diferentes localizaciones a lo largo del mundo.

El origen de estas tecnologías puede encontrarse en el año 2004, cuando Google publica un trabajo donde presenta el modelo de programación MapReduce. A partir de ahí, y con el surgimiento de Apache Hadoop como implementación *Open Source* de dicho modelo, el crecimiento de este tipo de tecnologías ha sido exponencial.

Por otro lado, en el ámbito de la Computación de Altas Prestaciones (High Performance Computing o HPC), existe una carrera entre empresas, instituciones y centros de investigación para entrar en la era de los supercomputadores exascale. Estos sistemas de computación deberán ser capaces de realizar 10^{18} operaciones en punto flotante por segundo, es decir, tener un rendimiento de 1 EXAFLOP. Hoy en día aún estamos lejos de alcanzar dicha meta, ya que el supercomputador que se alza con el primer puesto en el TOP500 en Junio de 2017 es capaz de alcanzar los nada despreciables 125,4 PETAFLOPS $(125,4 \times 10^{15} FLOPS)$, pero todavía falta un orden de magnitud para alcanzar el EXAFLOP, lo que supone un enorme salto

tecnológico.

Para llegar a alcanzar el EXAFLOP de rendimiento, los supercomputadores necesitarán que el envío de datos se realice de un modo rápido y eficiente, tanto dentro de un mismo nodo como entre nodos diferentes. Esta es una tarea difícil de lograr en los grandes supercomputadores, así como en los programas con una alta demanda computacional, como los provenientes de problemas científicos y de análisis de datos. Además, las *Application Programming Interfaces* (APIs) deberán proveer al programador de métodos que le permitan llevar a cabo la explotación de cantidades excepcionales de paralelismo y, al mismo tiempo, hacerlo de modo que la facilidad de uso y la programabilidad no sea un problema, así como soportar arquitecturas heterogéneas, tales como las que incorporan GPGPUs o sistemas manycore. Otro requisito es el de dar soporte a mecanismos de tolerancia a fallos, mediante los cuales una aplicación se podría recuperar de un fallo software o hardware, para continuar con la ejecución normal del proceso justo desde el punto donde se produjo dicho error.

Las APIs de los lenguajes de programación de computación paralela clásicos (como por ejemplo MPI, OpenMP, etc) se encuentran en una etapa de desarrollo y mejora, con el objetivo de alcanzar los requisitos mencionados anteriormente. Por otro lado, los entornos de desarrollo Big Data (por ejemplo, Spark o Hadoop) ya cumplen con algunas de estas características, como la tolerancia a fallos o la facilidad de programación. Aún así, todavía no está claro que paradigma encaja mejor para alcanzar un alto rendimiento y, al mismo tiempo, manejar grandes cantidades de datos de una forma eficiente en un amplio rango de códigos científicos.

Además de lo expuesto en el párrafo anterior, existe una cierta tensión entre la necesidad de reducir el movimiento de datos y el potencial de organizar y ejecutar tareas de un modo dinámico (con el movimiento de datos que ello implica). El rol del usuario al tratar de balancear dichos parámetros es todavía un punto de debate. Aún así, podríamos preguntarnos, ¿existe una diferencia fundamental entre HPC y Big Data?, ¿o la diferencia simplemente reside en las aplicaciones y el software empleado? Mientras que la Computación de Altas Prestaciones se centra más en grandes cargas computacionales, las tecnologías Big Data tienen como objetivo aplicaciones que necesitan manejar grandes y complejos conjuntos de datos. Dichos conjuntos de datos son, habitualmente, del orden de varios PebiBytes o TebiBytes de tamaño.

A primera vista, estas diferencias entre Big Data y HPC pueden resultar extrañas, ya que, en el fondo, ambos ecosistemas engloban tecnologías que permiten realizar tareas de cómputo en paralelo con la consiguiente reducción de tiempos de ejecución y mejora del rendimiento.

Entonces, ¿a qué se deben estas diferencias? Podemos destacar varios factores. El primero es que, con el surgimiento de las tecnologías Big Data, también ha surgido todo un ecosistema de nuevas aplicaciones asociadas a ellas (schedulers, sistemas de fichero paralelos, etc). Estas aplicaciones ya tenían anteriormente sus equivalentes en el mundo HPC. La gran diferencia es que las nuevas tecnologías incorporan funcionalidades que son necesarias para implementar algunas de las características típicas de las aplicaciones Big Data (por ejemplo, la tolerancia a fallos). La consecuencia más inmediata y evidente de esta divergencia es la existencia de dos ecosistemas paralelos y completamente diferentes e incompatibles (al menos de momento) entre el mundo HPC y en el mundo Big Data.

Otro de los factores causantes de dicha divergencia es, aunque pueda parecer poco importante en primera instancia, el lenguaje de programación empleado. Normalmente los lenguajes de programación con más aceptación en el mundo HPC son C, C++ o Fortran. Mientras, las tecnologías Big Data suelen utilizar lenguajes de más alto nivel, como por ejemplo, Java, Python o Scala, ya que estos lenguajes ofrecen una mejor programabilidad. Esto resulta ser un problema, ya que las aplicaciones o librerías tradicionalmente empleadas en HPC, o incluso en otras áreas científicas, no están implementadas mediante los lenguajes que se están usando en Big Data. Las tecnologías Big Data se han orientado, desde su inicio, hacia su uso por parte de los denominados "científicos de datos", más preocupados por el tratamiento estadístico de los datos que por las características a bajo nivel de sus aplicaciones. Por lo tanto, tienden a usar lenguajes de más alto nivel como los citados anteriormente, o incluso de propósito específico, como por ejemplo SQL o R, para desarrollar códigos de una forma rápida.

Estos dos factores son, sin duda, una dificultad en el camino hacia los supercomputadores exascale. Varias investigaciones en el área del HPC ya han manifestado la conveniencia de que los dos mundos, HPC y Big Data, han de converger para poder alcanzar este objetivo. De momento, sin embargo, no se han propuesto metodologías o tecnologías que hagan que los dos mundos coexistan o converjan.

A medida que las investigaciones científicas demanden una mayor velocidad de cómputo y capacidad para analizar datos, la potencial interoperabilidad de estos dos mundos es crucial para el futuro. En esta tesis, se emplean tecnologías Big Data para tratar problemas científicos que son computacionalmente intensivos en cuanto a tiempo de ejecución (típico en problemas HPC) y, al mismo tiempo, tienen un gran tamaño en cuanto a datos de entrada (típico en problemas Big Data), con el objetivo de mejorar el tiempo de ejecución, la escalabilidad y la eficiencia.

Dichos problemas científicos abordados en esta tesis tienen una serie de características comunes que los hacen adecuados para demostrar los beneficios generados por la sinergia entre los mundos HPC y Big Data:

- Requerir una gran cantidad de datos de entrada.
- Tener una elevada carga computacional, ya que las aplicaciones HPC son computacionalmente intensivas.
- Deben generar una gran cantidad de datos de salida. En muchos casos, aplicaciones de ambos mundos forman parte de pipelines de trabajo que proporcionan datos de entrada para otras aplicaciones. Por lo tanto, la salida de dichas aplicaciones también suele generar una gran cantidad de información.
- Deben estar escritos, al menos en su mayor parte, en un lenguaje de programación típico de las aplicaciones HPC.

Teniendo en cuenta dichos requisitos, los problemas científicos considerados en esta tesis se engloban dentro de las áreas científicas de la **Genómica** y del **Procesamiento del Lenguage Natural (PLN)**.

El área del **Procesamiento del Lenguaje Natural (PLN)**, está considerada como una de las metodologías más apropiadas para poder estructurar y organizar la información textual accesible a través de Internet. El procesamiento lingüístico de grandes cantidades de texto es una tarea compleja que requiere del uso de varias subtareas organizadas en módulos interrelacionados. Estos módulos son necesarios para poder llevar a cabo tareas más complejas, como la traducción automática, la recuperación de información o sistemas de vigilancia tecnológica. Generalmente, se necesita que el procesamiento lingüístico en tareas NLP sea lo más precisa y eficiente posible.

Uno de los mayores problemas que presentan los módulos PLN es su alto coste computacional y sus dificultades de escalabilidad. Esto los hace inviables para el análisis de grandes volúmenes de documentos (GibiBytes e incluso TebiBytes). De este modo, en esta tesis hemos considerado que el uso de soluciones HPC y Big Data se hace indispensable si se quiere reducir de forma notable los tiempos de cómputo, mejorar la escalabilidad del sistema y abordar problemas de un tamaño aún mayor. El conjunto de módulos PLN que se han seleccionado han sido los que permiten realizar la identificación y clasificación de entidades con nombre o NERC por sus siglas en inglés *Named Entity Recognition and Classification*.

El NERC es una línea de investigación en sí misma dentro del Procesamiento de Lenguaje Natural. El objetivo de esta línea es reconocer, identificar y clasificar nombres dentro de un texto. Este proceso se lleva a cabo con la ayuda de una lista predeterminada de categorías, por ejemplo, Persona, Lugar, Organización, etc. Las herramientas del estado del arte en esta área usan tanto técnicas lingüísticas basadas en gramáticas como modelos estadísticos. Normalmente los sistemas basados en gramáticas hechas a mano obtienen mejor precisión, pero con el coste de meses de trabajo de expertos lingüistas con experiencia en computación. Por otro lado, los sistemas NERC estadísticos requieren de una gran cantidad de datos de entrenamiento anotados a mano. También existen los enfoques semi-supervisados, que han sido sugeridos para evitar parte del esfuerzo de anotación.

Todos estos enfoques adolecen de la misma característica, que es la capacidad de procesar grandes cantidades de datos en un tiempo razonable. Por ejemplo, procesar toda la Wikipedia en español conlleva un tiempo de ejecución de alrededor de 19 días con un sistema NERC del estado del arte incluido en el repositorio Linguakit[1]. Los módulos PLN de dicho sistema están implementados en lenguaje Perl, y funcionan como un pipeline de trabajo.

El lenguaje Perl se usa frecuentemente en tareas de PLN, ya que está preparado para tratar con datos de tipo texto, debido sobre todo a su potencia en el uso de expresiones regulares. Sin embargo, el caso de emplear herramientas escritas en lenguaje Perl o en cualquier otro lenguaje diferente de Java con Apache Hadoop no está contemplado. Para poder realizar dicha tarea, Apache Hadoop permite emplear la herramienta Hadoop Streaming. Dicha herramienta permite al usuario ejecutar programas que siguen el modelo MapReduce empleando cualquier lenguaje de programación, con la única limitación de que el código debe leer los datos de entrada desde la entrada estándar, o *stdin*, y escribir los datos de salida en la salida estándar, o *stdout*. El problema es que dicha herramienta ha demostrado carecer de la eficiencia esperada. Debido a ello en esta tesis se han explorado otro tipo de soluciones a este problema, el cual también podría presentarse en otro tipo de aplicaciones.

Tras explorar dichas soluciones, se ha creado la herramienta **Perldoop**, que permite traducir códigos escritos en lenguaje Perl a códigos escritos en Java preparados para su ejecución con Apache Hadoop sin necesidad del módulo de Hadoop Streaming. Esto permite que investigadores en PLN puedan traducir y ejecutar sus códigos desarrollados en Perl en un entorno Big Data, con la consiguiente mejor eficiencia posible, escalando adecuadamente y, además, con la tolerancia a fallos que proporciona.

[1] https://linguakit.com

El proceso de traducción es relativamente sencillo, ya que tan sólo requiere por parte del usuario añadir una serie de *tags* o etiquetas en el código Perl y emplear unos *templates* o plantillas en lenguaje Java que ya provee la propia herramienta Perldoop. De este modo, los investigadores en el área del PLN no tienen que profundizar en un nuevo lenguaje o unas nuevas tecnologías con las que no tienen por qué estar familiarizados. Todo este proceso de traducción, así como de la implementación y resultados obtenidos, se tratan con profundidad en el Capítulo 2. Así mismo, también se muestran ejemplos de traducción sencillos, de modo que el usuario pueda comprender fácilmente el proceso.

Para obtener resultados de un problema real y con un software del estado del arte, en dicho capítulo se ha empleado Perldoop con los modulos NERC de Linguakit. La plataforma seleccionada para llevar a cabo los experimentos ha sido un clúster Big Data del Centro de Supercomputación de Galicia. Con estas características, los resultados han mostrado un speed-up de 57.9× usando 64 cores para los módulos NERC traducidos, frente al speed-up de 29.4× obtenido por Hadoop Streaming. Como se puede apreciar, el rendimiento de los módulos usando Perldoop es aproximadamente el doble que usando la herramienta Hadoop Streaming.

La segunda área científica que tratamos en esta tesis es la **Genómica**. El primer paso en un análisis genómico de ADN o ARN es siempre la lectura de una muestra biológica, para poder, de este modo, trasladar la información contenida en dicha muestra desde el entorno biológico al computador, y así poder analizar dichos datos computacionalmente. Esto se consigue mediante el uso de las tecnologías de ultrasecuenciación. Gracias a estas tecnologías, se han desarrollado máquinas que permiten introducir la muestra y obtener directamente los datos en un formato digital, es decir, después del proceso de ultrasecuenciación, los datos tienen forma de uno o varios ficheros de texto que siguen un determinado formato.

En los últimos años, dichas tecnologías de ultrasecuenciación han dado un salto importantísimo en lo referente a la cantidad de datos que se pueden obtener de muestras, así como en la velocidad a la que se pueden obtener dichos datos. Dicho salto en lo referente a estas tecnologías es en parte debido a los avances científicos llevados a cabo por compañías como, por ejemplo, Illumina, o a centros como el Wellcome Trust Sanger Institute del Reino Unido.

Debido a la velocidad en la obtención de datos y a la cantidad obtenida de los mismos, éstos pueden alcanzar fácilmente el orden de varios GibiBytes o TebiBytes de datos, dependiendo del tipo de muestra. El problema que se presenta ahora es dar sentido científico a dichos datos, ya que la velocidad a la que aumenta su tamaño está sobrepasando a la velocidad a la que crece la capacidad de cómputo de un procesador.

1	2	3	4	5	6
AACGT- ACCGTT	-AACGT ACCGTT	A-ACGT ACCGTT	AACGT—— ——ACCGTT	AA-CGT- A-CCGTT	-A-A-C-G-T- A-C-C-G-T-T

Tabla 1: Seis posible alineamientos para las secuencias de ejemplo.

Normalmente, los profesionales en el área emplean flujos de trabajo descritos en las *GATK Best Practices* del Broad Institute, del MIT y Harvard para dar sentido científico, o analizar, los datos obtenidos. Dichas buenas prácticas describen los pasos a seguir, así como el software a utilizar en cada uno de esos pasos, para elaborar una serie de análisis concretos. Algunos de estos pasos son comunes a varios de los flujos de trabajo.

Una de las etapas que son comunes a varios de estos flujos de trabajo es la del alineamiento o mapeado de secuencias. Para explicar lo que es el alineamiento, primero hay que tener en cuenta que las secuencias de ADN o proteínas pueden cambiar su codificación mientras evolucionan en el tiempo. Los tipos más simples de mutaciones son mutaciones puntuales e inserciones/extracciones, también conocidos como *indels* (insertion/deletion). El alineamiento trata de identificar estas mutaciones mediante algoritmos de alineamiento clásicos. Por ejemplo, al alinear las dos secuencias de ADN, *AACGT* y *ACCGTT*, algunos de los posibles resultados son los que se pueden ver en la Tabla 1, donde cada carácter representa una base nitrogenada y el guión representa un hueco.

La cuestión ahora sería saber cuál de dichos alineamientos es el "mejor". Para ello, existen varios algoritmos de puntuación o *score* explicados en la literatura. Las referencias a la literatura donde se explican dichos algoritmos de puntuación se indican en la Introducción de este documento. Sin embargo, y teniendo en cuenta que hay varios posibles alineamientos y que es necesario puntuarlos, y que al mismo tiempo estamos hablando de una gran cantidad de datos, este paso es uno de los más costosos computacionalmente dentro de los flujos de trabajo mencionados. Al mismo tiempo, es uno de los pasos más fundamentales, por lo tanto, obtener herramientas que permitan realizar este alineamiento de una forma eficiente y escalable es un requisito indispensable.

Dentro de las herramientas del estado del arte, existen varias que permiten llevar a cabo el alineamiento de secuencias. En concreto, en el caso de alineamiento de secuencias cortas (de menos de 100 caracteres o pares de bases), la herramienta BWA (Burrows-Wheeler Aligner) es una de las más utilizadas, como así lo refleja el hecho de que es la indicada en las *GATK Best Practices* para realizar la fase de alineamiento. Dicha herramienta está implementada

en C y utiliza paralelismo a nivel de hilo (es decir, en memoria compartida). Debido a su amplia aceptación entre la comunidad de investigadores en Bioinformática y a los detalles de su implementación, se ha seleccionado esta herramienta como caso de estudio en esta tesis. A pesar de disponer de una versión paralela para sistemas de memoria compartida, el alineamiento con BWA puede suponer varios días de cómputo dependiendo del tamaño de los datos de entrada.

Debido a lo expuesto en el párrafo anterior, y al tratar con una gran cantidad de datos de entrada proveniente de las máquinas de ultrasecuenciación, las tecnologías Big Data parecen las adecuadas para tratar con este problema. Al mismo tiempo, al tratarse de un problema computacionalmente intensivo, las tecnologías HPC también podrían ser adecuadas. Por ello, se ha implementado la herramienta **BigBWA**, la cual emplea Hadoop como tecnología Big Data para poder realizar en paralelo el alineamiento, mientras que internamente emplea los algoritmos implementados en BWA. Dichos algoritmos están escritos en lenguaje C, y se hace uso de ellos mediante llamadas desde Java en las partes que son computacionalmente intensivas mediante el uso de JNI (*Java Native Interface*). Debido a que las secuencias de entrada son totalmente independientes entre sí, estamos hablando de un problema de los denominados *Embarrassingly Parallel*, ya que los datos se dividen en fragmentos y cada uno de dichos fragmentos pueden ser procesados independientemente de los demás.

Siguiendo esta estrategia, cuyos detalles se pueden ver en el Capítulo 3, y empleando un clúster Big Data desplegado en Amazon Web Services se han llevado a cabo una serie de experimentos que demuestran la viabilidad de esta herramienta. Los datos de entrada se han tomado de un repositorio público de secuencias de ADN humano, que proviene del proyecto *1000 Genomes*. Los resultados de dichos experimentos han mostrado como BigBWA consigue una aceleración de 36.6× empleando 64 cores con respecto a la versión secuencial de BWA (1 thread), mientras que, al mismo tiempo, es más rápido que otras herramientas similares del estado del arte. Además incorpora características típicas de los ecosistemas Big Data, como la tolerancia a fallos.

Dentro del mundo de las tecnologías Big Data, los avances se producen muy rápidamente. Así lo demuestra la aparición de Apache Spark, un motor que mejora Apache Hadoop en varios aspectos. Por ejemplo, permite ir más allá del modelo MapReduce o proporciona mejoras en el acceso a disco. Debido a ello, en esta tesis se ha implementado también una versión de BWA empleando Apache Spark, surgiendo así **SparkBWA**.

SparkBWA sigue la misma estrategia que su predecesor, BigBWA. Es decir, realiza la di-

visión de datos y tareas empleando tecnologías Big Data, en este caso Apache Spark, mientras que el cómputo intensivo requerido por el algoritmo de alineamiento se realiza con el código en C de BWA mediante JNI. De este modo, las ventajas obtenidas en BigBWA se mantienen y, al mismo tiempo, se incluyen mejoras que proporciona Apache Spark. Por poner un ejemplo, una fase de preprocesado de datos que requería BigBWA, ahora con SparkBWA se realiza de forma más eficiente empleando funciones nativas de Spark.

Todos los detalles de la implementación, así como las mejoras mencionadas en el párrafo anterior, se pueden ver en el Capítulo 4. En dicho capítulo también se presentan los resultados obtenidos. SparkBWA consigue una aceleración de hasta 85.6× usando 128 cores con respecto a la versión secuencial de BWA (1 thread), mientras que BigBWA y otras herramientas del estado del arte se quedan alrededor de 66× con el mismo número de cores y también con respecto a la versión secuencial de BWA (1 thread).

Uno de los objetivos de la comparación de secuencias de proteínas es descubrir similaridades estructurales o funcionales entre proteínas. Biológicamente, proteínas similares podrían no mostrar una similaridad clara. Por ejemplo, si la similitud de las secuencias es baja, el alineamiento de un par de secuencias puede fallar al identificar secuencias biológicamente relacionadas. Sin embargo, la comparación simultánea de varias secuencias a menudo permite encontrar similitudes que son imposibles de identificar en el caso de un par de secuencias. Dicho es lo que se conoce como *Multiple Sequence Alignment* o MSA, en el cual se pretende alinear varias secuencias al mismo tiempo. La mayor parte de implementaciones de algoritmos que llevan a cabo MSA llevan a problemas de optimización de combinatoria NP completos. Como parte de esta tesis también se ha llevado a cabo un trabajo en el que se integra un software para MSA, denominado PASTA, en un entorno Big Data, surgiendo así la herramienta **PASTASpark**.

En este caso, PASTA engloba diversas herramientas para llevar a cabo el alineamiento múltiple. Dichas herramientas están implementadas en diferentes lenguajes de programación, como por ejemplo, Python, e incluso se realizan llamadas a binarios ejecutables, por lo que es otro candidato ideal para este nuestros objetivos en esta tesis. Cada una de dichas herramientas se utiliza en diversas fases. En este caso, el trabajo se ha centrado en paralelizar una fase en la que interviene el alineador conocido como MAFFT.

PASTA incluye por defecto un modo paralelo, en el que se ejecutan varios procesos al mismo tiempo, con la característica de que dichos procesos sólo pueden ser ejecutados en la misma máquina. Sin embargo, con PASTASpark, la fase donde interviene MAFFT, que es la

más costosa computacionalmente, se ejecuta en un cluster de computación empleando Apache Spark. Los detalles de nuestra propuesta se muestran en el Capítulo 5. De todos modos, cabe mencionar que en este caso una de las fases de PASTA supone un cuello de botella que no permite a PASTASpark escalar de manera adecuada. Debido a eso, los resultados se ven limitados por la ley de Amdahl, aunque están muy próximos a dicho límite. Por ejemplo, empleando un cluster en Amazon Web Services, con el conjuntos de datos de entrada más grande del que se dispone, se baja de las 24 horas de ejecución, mientras que usando la versión original de PASTA se necesitan varios días y no se consigue completar la ejecución.

Para finalizar este resumen, mencionar que las conclusiones globales obtenidas de todos los trabajos aquí expuestos se encuentran detalladas en el Capítulo 6.

Contents

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

CHAPTER 1

INTRODUCTION

1.1. Motivation

The human being has entered in the era of data. According to IBM, every day 2.5 quintillion bytes of data are created, in such a way that 90% of the data all over the world were created just only in the last two years [1]. This data comes from everywhere: sensors that are dedicated to gather climate information, posts to social networks, digital pictures and videos, smartphones GPS signals, etc. But the Internet of Things or the Social Networks are not the only responsible of this data explosion, science also plays an important role. As an example, the Large Hadron Collider (LHC) in Geneve (Switzerland), generates about 30 PetaBytes of data per year [2], and in its first phase, the Square Kilometre Array (SKA) radio telescope will produce 160 TeraBytes of raw data per second [3]. All this data is what is known as **Big Data**.

The task of processing and generating knowledge from this huge amount of information can be an unaffordable problem. We can summarize the main challenges to deal with Big Data in the following way:

- Data is usually not structured.
- The computing time to process and access this information is typically very high.
- The memory consumed by scientific applications that make use of this data exceeds by far the memory that is typically installed on a workstation.

Name	Description	Performance
`Sunway TaihuLight`	National Supercomputing Center in Wuxi (China). Supercomputer at the top of the TOP500 list (June 2017).	125.4 PFLOPS ($\times 10^{15} FLOPS$) [7]
`FinisTerrae II`	Galicia Supercomputing Centre (CESGA) cluster.	328 TFLOPS ($\times 10^{12} FLOPS$) [8]
`PlayStation 4`	Videogame console by Sony.	1.8 TFLOPS ($\times 10^{12} FLOPS$) [9]
`Nexus 5`	Android mobile phone.	18 GFLOPS ($\times 10^{9} FLOPS$) [9]

Table 1.1: Comparing processing power of different platforms.

To overcome these limitations, Google developed the **MapReduce** model [4, 5]. MapReduce is a programming model that comes with an associated implementation for processing and generating big data sets on commodity clusters. In this model, the runtime takes care of:

- Partitioning the input data.
- Scheduling the program's execution across a set of machines.
- Handling machine failures.
- Managing inter-machine communications.

Since the release of the MapReduce model and the birth of its *Open Source* implementation, Apache Hadoop [6], the growth of Big Data technologies has been exponential.

On the other hand, nowadays, the High Performance Computing (HPC) community is involved in a race between companies, institutions, and research centres to reach the Exascale milestone. Exascale refers to supercomputers capable of executing 10^{18} floating point operations per second, this is, one EXAFLOP per second. To give some perspective to this number, some comparison data is displayed in Table 1.1. As the numbers reveal, there is still an order of magnitude to reach Exascale for the first supercomputer at the TOP500 [7] list.

To reach that performance, future supercomputers will need that data delivery to be fast and efficient, both from memory and disk, and also across the network and between processors. This is a difficult task to achieve in big supercomputers, and also in large computations, like those present in scientific and data analytics problems. Also, Exascale Application Programming Interfaces (APIs) will need to make easy for the programmer the exploitation

of exceptional amounts of parallelism in applications, enabling the processing of significant amounts of data, and supporting several different architectures, including those based upon heterogeneous cores or accelerators. These APIs and their implementations will need to carefully manage different kinds of memories within each node. Moreover, the need to conserve energy has led to an increased focus on reducing data motion at all levels of the memory hierarchy, from low cache levels to main memory, requiring a rethinking of algorithms as well as of the entire HPC software stack. In addition, Exascale execution software systems will need to ensure that jobs continue to run despite the occurrence of system failures and other kinds of hardware or software errors.

Traditional programming interfaces for expressing parallelism (in particular, MPI [10], OpenMP [11] and OpenACC [12]) are being further developed to achieve some of these requirements. On the other hand, Big Data frameworks (such as Apache Spark [13] or Apache Hadoop [6]) have already implemented fault tolerance and programmability requirements. However, it is still unclear which paradigm is a natural fit for expressing computations and handling data in a broad range of scientific application codes.

At the same time, there is a tension between the need to reduce data motion and the potential to dynamically schedule and execute tasks. The role of the user in balancing these is also still being debated. Yet, is there a fundamental difference between HPC and Big Data or does the difference only reside in the application and software usage? While HPC mostly focuses on large computational loads, Big Data targets applications that need to handle very large and complex data sets. These data sets are typically of the order of multiple terabytes or exabytes in size. Big Data applications are thus very demanding in terms of storage, to accommodate such a massive amount of data, while HPC is usually thought more in terms of pure computational needs [14].

At this point, and after this explanation, a comparison between the Big Data and HPC stacks is needed. A stack based on [15] can be seen in Figure 1.1. In this figure, we can observe how both ecosystems share some attributes, for example, a notable reliance on open source software and the x86 hardware. However, as scientific research increasingly depends on both high-speed computing and data analytics, the potential interoperability of these two ecosystems is crucial for the future. Also, despite the similarities, they differ markedly in their foci and technical approaches.

Regarding the HPC ecosystem, commodity clusters (Intel/AMD) and purpose built processors (IBM's BlueGene) that dominated the previous decade, have been augmented with

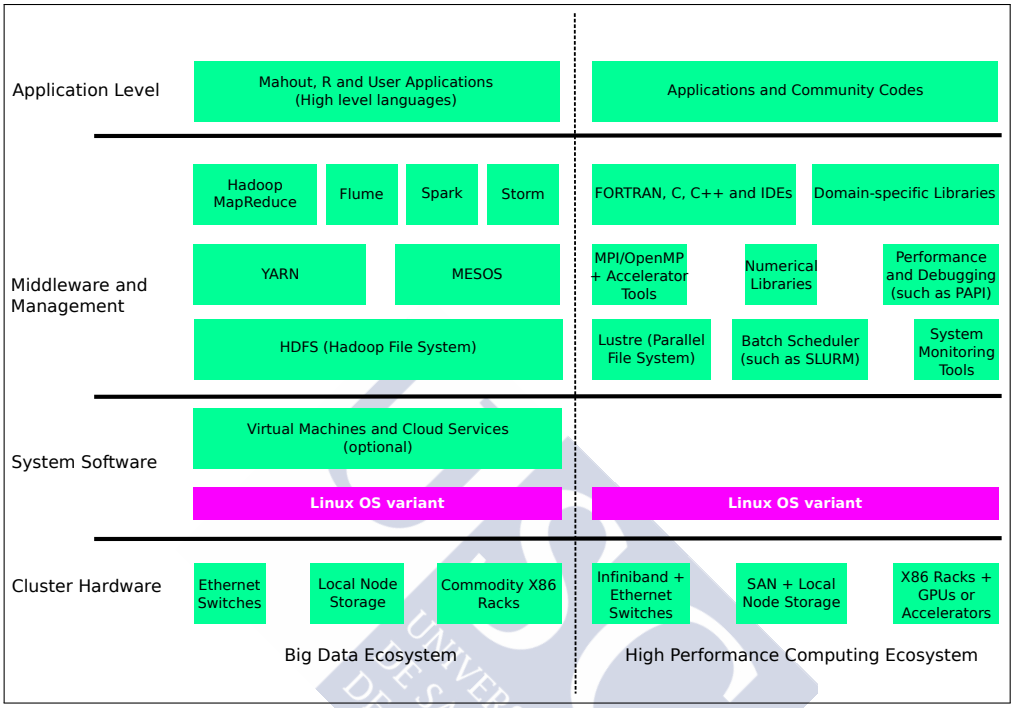

Figure 1.1: Stack comparison between Big Data and HPC ecosystems.

computational accelerators in the form of coprocessors, graphical processing units (GPUs) or FPGAs. They also include high-speed low-latency networks (such as Infiniband) and Storage Area Networks (SANs). This hardware ecosystem is optimized for performance, rather than for minimal cost or energy consumption.

On top of the hardware, Linux provides the operating system, augmented with parallel file systems (Lustre [16]) and batch schedulers (Torque [17] or SLURM [18]) for parallel job management. MPI [10] and OpenMP [11] are used for internode and intranode parallelism, augmented with CUDA [19] or OpenCL [20]. Numerical and domain specific libraries such as LAPACK [21] and PETSc [22] complete the software stack. Applications are typically written in Fortran, C or C++.

On the other hand, in the Big Data Ecosystem, clusters are typically based on Ethernet networks and local storage, which focus in cost and capacity. In this case, on top of the Linux operating system, HDFS [23] is commonly used, completed with YARN [24] or Mesos [25]

for job scheduling. In this case, on top of the schedulers, different technologies can be found, such as Spark or Hadoop MapReduce. Spark includes some libraries for Machine Learning or Graphs Processing, with some aspects related to numerical computation but more oriented to statistics than matrix algebra or simulation. Finally, on top of the Big Data stack, user applications are usually written in high level languages (Perl or Python, for example). This is another difference, since HPC programs are written in low level languages, which are better suited for performance optimization.

Despite the fact that both ecosystems share some ideas, philosophy, and even technology, they are very different. In addition, the barrier between Big Data technologies and classic High Performance Computing applications is still not clear. In this thesis, we use Big Data technologies to deal with some scientific problems that are computationally intensive regarding execution time (typical in HPC problems) and, at the same time, have a large input data size (typical in Big Data), with the objective of improving their execution time, scalability and efficiency. Conclusions derived from this work can clarify where this barrier stands, or even prove if this barrier exists at all, and maybe help to solve the key question that in recent times has arisen among the HPC community: should Big Data be considered part of the High Performance Computing field?

1.2. The MapReduce programming model: Apache Hadoop

Next, a brief overview of how the MapReduce programming model and Apache Hadoop work is introduced. In the MapReduce model the computation takes a set of *input key/value pairs*, and produces a set of *output key/value pairs*. The model expresses the computation as two functions written by the user: **map** and **reduce**.

The **Map** function takes as input a key/value pair and produces an intermediate list of key/value pairs. The function is called one time per each input key/value pair in the input data. Then, the intermediate values associated with the same key are grouped together. Sometimes, data need to be redistributed according to the intermediate keys, in a way that all the values belonging to the same key are in the same computing node. This is what is known as the *shuffle* phase. After that, data reaches the reduce function.

The **reduce** function, accepts an intermediate key and a list of values for that key. It operates with these values to conform a set of output data. The intermediate values are supplied to the user's reduce function via an iterator. This allows the user to handle lists of values that

are too large to fit in memory.

By using these two functions from the model, the user should be able to process this large amount of input data. The map invocations are distributed across multiple machines by automatically partitioning the input data into a set of splits. The input splits can be processed in parallel by different machines, or be processed in the same machine. Reduce invocations are distributed by partitioning the intermediate key space into pieces using a partitioning function.

From this model, the best known Open Source implementation is Apache Hadoop [6]. Apache Hadoop is a framework used for distributed storage and processing of big data sets. It is written in Java with some native code in C, and it consists mainly of three parts:

1. **HDFS**: Distributed file system that stores data on local disk of commodity hardware.

2. **YARN**: Resource management platform responsible for managing computing resources in clusters, using them for scheduling users applications.

3. **Hadoop MapReduce**: The implementation of the MapReduce programming model.

Hadoop is mainly known because of the MapReduce model and its distributed filesystem (HDFS). However, the name is also used for a family of related projects that fall under the umbrella of its infrastructure for distributed computing and large-scale data processing [26]. This is the so called Hadoop ecosystem. Some examples of these projects are, apart from the already mentioned HDFS, YARN and Hadoop MapReduce: Apache Pig [27], Apache Hive [28], HBase [29], etc. However, all of these projects rely on YARN and the HDFS.

The Hadoop Distributed File System **(HDFS)** is designed to store very large data sets with reliability, and to stream those data sets with a high bandwidth to user applications [23]. HDFS stores the file system metadata separated from the data itself and is rack aware. As in other distributed file systems, like PVFS [30], Lustre [16], or GFS [31, 32], HDFS stores metadata on one or more dedicated servers, called the NameNodes. Application data are stored on other machines, called DataNodes. However, a NameNode process and a DataNode can coexist in the same machine. All servers are fully connected and communicate with each other using TCP-based protocols. The data stored in the DataNodes is divided in blocks (0 to N blocks per DataNode) of a certain size (typically 128 or 64 MB) and with a certain replication factor (the default is 3).

An example of how HDFS works can be seen in Figure 1.2. In this example there are 6 DataNodes and one NameNode. There are 3 files in HDFS (represented in white, red and

blue), each one of them occupying two blocks, with a replication factor of three. Considering this configuration, for example, block 1 of the blue file is stored in the first, third and sixth DataNode, while block two of the same file is stored in DataNodes two, four and five.

Regarding **YARN** (Yet Another Resource Negociator), it is the Hadoop cluster resource management system. It was introduced in Hadoop 2 to improve the MapReduce implementation, but it is general enough to support other distributed computing paradigms [24].

YARN has a flexible model for making resource requests. A request for a set of containers (processes in the Hadoop terminology) can have different parameters defining the amount of computer resources required for each container (memory and CPU), as well as locality constraints for the containers in that request. It provides its services by using two types of daemons: a Resource Manager (one per cluster) to manage the use of resources across the cluster, and Node Managers running on all the nodes in the cluster (one per node) to launch and monitor containers. A container executes an application specific process with a constrained set of resources. Depending on how YARN is configured, a container may be a Unix process or a Linux cgroup. Figure 1.3 is a modified figure from [6], and illustrates how YARN runs an application.

To execute an application on YARN, a client contacts the Resource Manager and requests it to run an Application Master Process (step 1 in Figure 1.3). The Resource Manager then finds a Node Manager that can launch the Application Master in a container (steps 2a and 2b). Precisely what the Application Master does once it is running depends on the application itself. It could simply run a computation in the container it is running in and return the result to the client. Or it could request more containers from the Resource Managers (step 3), and use them to run a distributed computation (steps 4a and 4b).

Finally, locality is critical in ensuring that distributed data processing algorithms use the

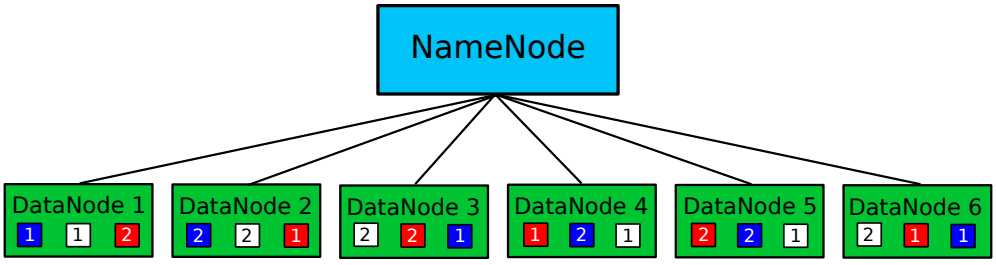

Figure 1.2: Example of how HDFS works.

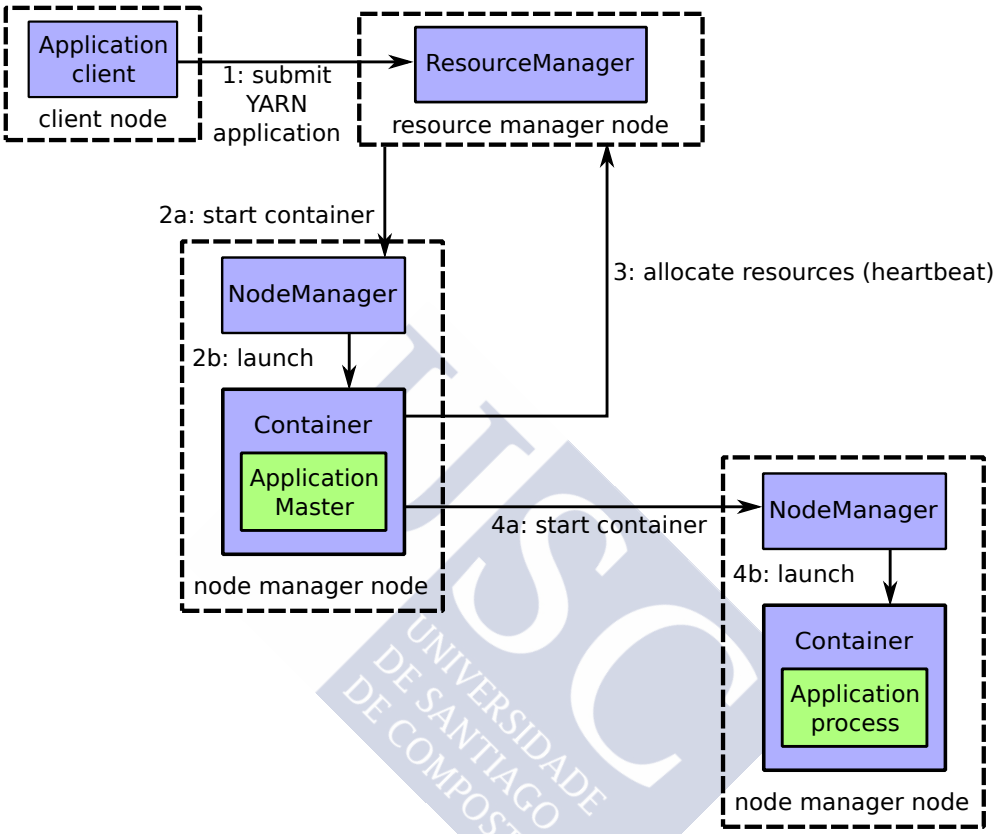

Figure 1.3: Example of how YARN works.

cluster bandwidth efficiently, so YARN allows an application to specify locality constraints for the containers it is requesting. Locality constraints can be used to request a container on a specific node or rack, or anywhere on the cluster (off-rack) [33].

1.3. Hadoop Limitations and Apache Spark

One of the main ideas from the HDFS is that the most efficient data processing pattern is a write-once, read-many-times pattern. For this reason, Hadoop shows good performance with embarrassingly parallel applications requiring a single MapReduce execution (assuming intermediate results between map and reduce phases are not huge), and even for applications

requiring a small number of sequential MapReduce executions [34]. Note that Hadoop can also efficiently handle jobs composed by one or more map functions by chaining several mappers followed by a reducer function and, optionally, zero or more map functions, saving the disk I/O cost between map phases. For more complex workflows, solutions as Apache Oozie [35] or Cascading [36], among others, should be used.

The main disadvantage of these workflow managers is the loss of performance when HDFS has to be used to store intermediate data. For example, an iterative algorithm can be expressed as a sequence of multiple MapReduce jobs. Since different MapReduce jobs cannot shared data directly, intermediate results have to be written to disk and read again from HDFS at the beginning of the next iteration, with the consequent reduction in performance. It is worth noting that even each iteration of the algorithm could consist of one or several MapReduce executions. In this case, the degradation in terms of performance is even more noticeable.

Apache Spark is a cluster computing framework designed to overcome the Hadoop limitations in order to support iterative jobs and interactive analytics, originally developed at University of California, Berkeley [13], now managed under the umbrella of the Apache Software Foundation. Spark extends the MapReduce model to efficiently support more types of computations (not only map and reduce), including batch applications, interactive queries, and streaming. One of the main features Spark offers in order to be able to perform this set of computations, is the ability to store the data in main memory between operations.

Another improvement with respect to Hadoop is the programming language. Programs developed to run with Hadoop are written in Java. Hadoop can also run programs written in other languages by using Hadoop Streaming, but some researchers have probed this tool to perform poorly [37]. On the contrary, Spark offers simple APIs in Python, Java, Scala, SQL, and even can be used from R. And not only that. It even includes a rich built-in libraries, for example, for Machine Learning, and also integrates closely with other Big Data tools. In particular, Spark can be run on Hadoop clusters and access any Hadoop data source [38].

The job topology is, however, very similar. A Spark application, at a high level, consists of a driver program that launches various parallel operations on a cluster. The driver program contains the application *main* function and defines **distributed datasets** on the cluster, then applies operations to them. These distributed datasets, commonly named RDDs or Resilient Distributed Dataset (RDD) [39], are one of the Spark main features. A RDD is simply an immutable distributed collection of objects. Each RDD is split into multiple partitions, which

may be computed on different nodes of the cluster. RDDs can contain any type of Python, Java, or Scala objects, including user defined classes. RDDs can be created in two ways: by loading an external dataset (for example, from HDFS), or by distributing a collection of objects (e.g., a list or set) in the driver program. Once created, RDDs offer two types of operations: *transformations* and *actions*.

Transformations construct a new RDD from a previous one. For example, one common transformation is filtering data that matches a predicate. **Actions**, on the other hand, compute a result based on an RDD, and either return it to the driver program or save it to an external storage system (e.g., HDFS). One example of an action is *first()*, which returns the first element in an RDD.

Transformations and actions are different because of the way Spark computes RDDs. Although the user can define new RDDs any time, Spark computes them only in a lazy way, that is, the first time they are used in an action. This approach might seem unusual at first, but makes a lot of sense when working with Big Data. For instance, consider an example where the user defined a text file and then filtered the lines that include "Python". If Spark were to load and store all the lines in the file as soon as possible, it would waste a lot of storage space, given that the user then immediately filter out many lines. Instead, once Spark sees the whole chain of transformations, it can compute just the data needed for its result. This is due the fact that, in Spark, all the operators in a job are used to construct a DAG (Directed Acyclic Graph). The DAG is optimized by rearranging and combining operators where possible

To run these kind of operations introduced in the previous paragraphs, driver programs typically manage a number of processes in the computing nodes called *executors*. An example of how the Driver Program and the Executors are distributed in Spark can be seen in Figure 1.4.

Finally, Spark's RDDs are by default recomputed each time the programmer runs an action on them. If an RDD is going to be reused in multiple actions, the programmer can ask Spark to persist it using the *persist* method. There are a number of different places where the data can be persisted, for example, memory, disk, memory and disk, etc. This place can be set as an option to the *persist* method. With the *persist* method and no option provided, Spark will store the RDD contents by default in main memory (partitioned across the machines in the computing cluster), and reuse them in future actions. The behaviour of not persisting by default may again seem unusual, but, it makes a lot of sense for big datasets: if the RDD is not going to be reused, there is no reason to waste storage space when Spark could instead stream

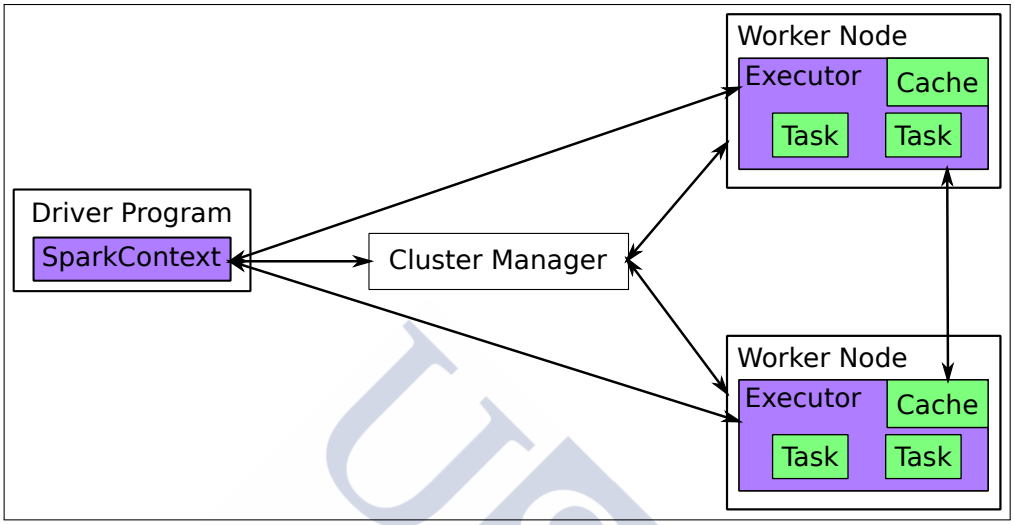

Figure 1.4: Example of how Spark works.

through the data once and just compute the result. These kind of strategies are also useful to implement fault tolerance in Spark. Thanks to the operations DAG, if the RDD fragments stored in one node are lost, they can be rebuild in another node by following the operations graph.

In conclusion, Spark overcomes most of the problems present in Hadoop. However, and despite all of its improvements and parallel philosophy, it is still not clear if Spark fits in the High Performance Computing world.

1.4. Case studies: Natural Language Processing and Genomics

To make progress in the approach between HPC and Big Data technologies, significant applications from the Genomics and Natural Language Processing fields will be designed and integrated into the Big Data frameworks, with the aim of boosting performance and improving scalability.

1.4.1. Natural Language Processing

Natural Language Processing (NLP) is a field of computer science, artificial intelligence and computational linguistics concerned with the interactions between computers and human (natural) languages. In particular, it is concerned with programming computers to fruitfully process large natural language corpora. It is considered as one of the more suitable methodologies to give a structure and organize the textual information accessible through the Internet. Linguistic processing of big quantities of text is a complex task that requires the use of several sub-tasks organized in various inter-related modules that run as a workflow. These modules are usually needed to carry out more complex tasks [40] such as machine translation.

A common task in NLP is the Named Entity Recognition and Classification (NERC). NERC is a research line inside NLP in consolidation phase. The objective of this line is to recognize, identify and classify names inside a text [41, 42, 43]. This process is performed with the help of a predetermined category list (for example, Person, Place, Organization, etc). NERC systems that use linguistic grammar-based techniques or statistical models, such as machine learning, are the state of the art tools. Grammar-based systems typically obtain better precision, but at the cost of lower recall and months of work by experienced computational linguists. Statistical NERC systems typically require a large amount of manually annotated training data. Semi-supervised approaches have been suggested to avoid part of the annotation effort [44, 45]. Many different classifier types have been used to perform machine-learned NERC, with conditional random fields being a typical choice [46].

In this thesis, Linguakit NLP modules have been selected as case study [41, 47, 48]. The NERC module in Linguakit consist of several modules in a pipeline. This NERC system from Linguakit can be observed in Figure 1.5. In these Figure, all the modules involved since the begin of the data process are shown. These modules are:

- **Basic analysis**

    1. Sentence Segmentation: This module divides the input in grammatical sentences.
    2. Tokenizer: Here the sentences are divided into tokens, this is, textual elements separated by white spaces or punctuation marks.
    3. Splitter: The splitter module uses linguistic information to divide those tokens that, despite of being various linguistic units, they are formally represented as an unique element. For example, contractions in spanish, "*del* = *de* + *el*".

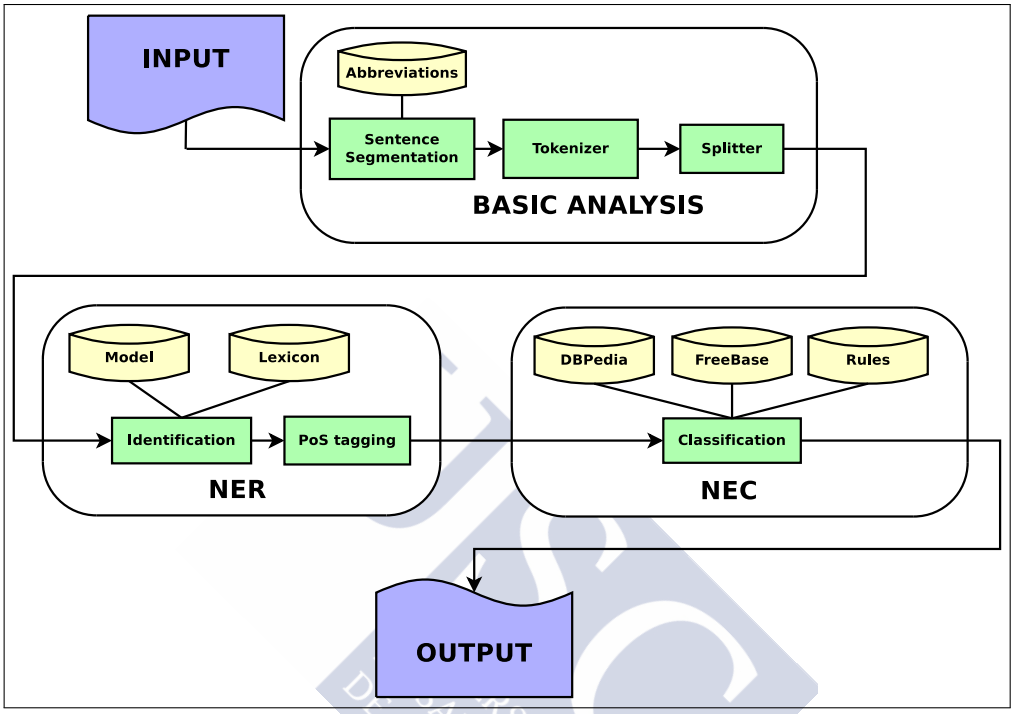

Figure 1.5: NERC system from Linguakit.

- **NER**: Named Entity Recognition

  1. Identification: This task consists of identifying as a single unit (or token) those words or chains of words denoting an entity, e.g. *New York, University of San Diego, Herbert von Karajan*, etc. The module is based on a set of language-independent rules that take into account information on both a large lexicon of forms and the relative position of words within the sentence.
  2. PoS tagging: This module assigns each token of the input text a single PoS tag provided with morphological information e.g. *singular and masculine adjective, past participle verb, plural and feminine noun*, etc. The module consists of a Bayesian classifier whose features are bigrams of tokens that represent the immediate left and right contexts of the target token.

- **NEC**: Named Entity Classification

Le	le	PP3CSD00
dije	decir	VMIS1S0
a	a	SPS00
María	maría	NP00SP0
el	el	DA0MS0
martes	martes	NCMN000
que	que	PR0CN000
me	me	PP1CS000
gusta	gustar	VMIP3S0
el	el	DA0MS0
cubismo	cubismo	NCMS000
.	.	Fp

Table 1.2: Linguakit NERC example.

1. Classification: The last step of the linguistic analysis is the semantic classification of those entities identified in the previous NER module. Named Entity Classification (NEC) is the process of classifying entities by means of classes such as "People", "Organizations", "Locations", or "Miscellaneous". NEC is a crucial task for several natural language applications, for example Question Answering and Information Extraction.

These modules identify and classify the named entities in two phases. First, the identification of token strings that joins to compose the entities (NER), and then the classification by using tags that characterize them in a semantic way. An output example of the Linguakit NERC tool is shown in Table 1.2 for the spanish sentence "*Le dije a María el martes que me gusta el cubismo*". In the example the first column contains the sentence tokens, the second column contains what are called lemmas and the third contains the unique tags which identify the tokens. For example, it identifies that "gusta" is a form of the verb "gustar" and assigns the corresponding tag "VMIP3S0", which starts with a "V" because it is a verb.

Despite all the different approaches that can be found in NERC systems from state of the art tools, all of them lack of the same feature. That is, the capacity to process huge quantities of text in reasonable time. For example, process the whole Wikipedia in Spanish language takes about 19 days (more than 450 hours) [49] when using the considered NERC system. The high execution time proves that one of the biggest problems of these techniques and tools is their high computational cost and scalability problems. For this reason, NLP modules are

non-viable for the analysis of big volumes (GigaBytes or TeraBytes) of documents. In this way, the use of High Performance Computing (HPC) is mandatory, in order to reduce the execution times, improve the system scalability and approaching bigger problems. However, state of the art implementations are written in languages that are not well suited for HPC such as Perl or Python. The reason is that these kind of languages are specially suited to work with text data, so are a good match to NLP. HPC oriented languages such as C or Fortran are more oriented to numerical data.

To overcome this limitation, Hadoop provides the Hadoop Streaming tool, which allows to run a program written in any programming language in Hadoop. However, as was stated before, this tool does not have a good performance [37]. For this reason, **Perldoop** [49] has been developed. The objective in this work was not to develop a powerful tool that allows to automatically translate any existent Perl code to Java, but a simple and easy-to-use tool that takes as input Perl codes written for Hadoop Streaming, following a reduced number of additional programming rules, and produces Hadoop-ready Java codes. In order to do that, Perldoop uses a system based on tags and templates. Templates is contain certain parts of the Java code that has no direct translation from Perl, such as class declarations, some auxiliary functions needed in Java or global variables. Regarding tags, they are used in the Perl code to indicate Perldoop to perform some type of specific translation or decision. The main benefit of using this methodology is the ease of use. Note that programmers have to insert tags in the Perl code and create the templates only once. After that procedure, the Perl code to be translated can be modified at any time. To obtain a new Java version of the code it is only necessary to execute Perldoop again, and it will be automatically generated.

A Hadoop cluster installed at the Galicia Supercomputing Center (CESGA), which consists of 64 nodes has been used in the experimental evaluation. With this infrastructure, an important reduction of the processing time is observed for all the parallel executions with respect to the sequential case both using Hadoop Streaming and Hadoop. For example, the original modules require about 19 days (more than 450 hours) to process the whole Wikipedia, while using Hadoop Streaming this time decreases to bout 16 hours. Despite these important improvements, the execution times using Hadoop Streaming are still high. However, when using the Hadoop-ready codes generated by Perldoop, this time is reduced to less than 2 hours. That is, $8\times$ faster than considering Hadoop Streaming.

More details about the design, implementation and Perldoop performance results can be found in Chapter 2.

1.4.2. Genomics

The first step in a DNA or RNA genomics analysis is always reading a biological sample, taking the genomic information from the sample into a digital format in a computer with the aim of analyzing this data. Emerging next-generation sequencing technologies (NGS) have broken many experimental barriers to genome scale sequencing, facilitating the extraction of huge quantities of sequences, which will further promote the future growth of biological databases. In the last years, the available biological data in a digital format has experimented a big and quick growth. Good examples are the DNA sequence information in the NCBI GenBank [50] database and the protein sequences in the UniProtKB/TrEMBL [51] database.

According to [52], compared to traditional Sanger capillary-based electrophoresis systems, NGS technologies provide ultra-high throughput with a two orders of magnitude lower unit data cost. However, they all share the common intrinsic characteristic of providing short read length, currently 25–75 base pairs (bp), this is, 25-75 characters, which is substantially shorter than the Sanger sequencing reads (500–1000 bp) [53].

These short reads, commonly named sequences, are composed of ASCII characters representing a nucleotide from the sequence, also known as nucleobases or simply, bases. For example, in the DNA case, we can find only four possible bases:

- A - Adenine
- C - Cytosine
- G - Guanine
- T - Thymine

Computer scientists and biomedical researchers face the challenge of transforming these short-reads into biological understanding. Consequently, bioinformatics tools need to be scalable; that is, they need to deal with an ever growing amount of data. Unfortunately, the amount of publicly available sequence data grows faster than the single core processor performance. Thus, modern bioinformatics tools need to take advantage of parallel computing [54].

To give meaning to all this data, according to the GATK [55] best practices from the Broad Institute [56], a wide number of DNA or RNA analysis pipelines perform a pre-processing raw data that comes from high throughput ultra sequencers. To be more precise, they say that "*pre-processing starts from raw sequence data and produces analysis-ready BAM files. This involves* ***alignment*** *to a reference genome*".

1	2	3	4	5	6
AACGT- ACCGTT	-AACGT ACCGTT	A-ACGT ACCGTT	AACGT------ -----ACCGTT	AA-CGT- A-CCGTT	-A-A-C-G-T- A-C-C-G-T-T

Table 1.3: Six different possible alignments for the example sequences.

This pre-processing or alignment is, basically, a way of arranging the sequences of DNA, RNA, or protein to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences [57]. This alignment of raw data to a reference genome is one of the most time consuming steps in every genomic analysis, and remains as a bottleneck in nowadays workflows Its computing time is very high in comparison with the other steps in the selected workflow. More information about the state of the art algorithms used to perform the alignment phase can be found in [58], [59] or [60].

Short reads alignment

DNA, RNA and protein sequences encode changes in evolution time through mutation. The simplest kinds of mutation are point mutations and insertions/deletions, also known as *indels*. These two types of change are modeled by classical alignment algorithms. As an illustrative example, a set of possible alignments of two DNA sequences, `AACGT` and `ACCGTT`, are shown in Table 1.3 by writing their bases on top of each other, in a way that, either two bases are paired, corresponding to presence or absence of a point mutation, or a base is paired with a gap, corresponding to an insertion or deletion. It is not allowed to pair a gap with a gap, not because two sequences cannot inherit the same deletion (they can), but because there is no enough information to infer such a course of evolution from just two sequences. In Table 1.3, six example alignments that can be generated following these rules are presented [59]. The question now is, which one of the alignments is the "best"? At first sight, alignment 1 may seem the best, as it has four matches, while alignment 2 only has one, alignment 3 has two, and alignments 4 and 6 have no matches. However alignment five also contains four matches, but it contains three gaps, while alignment 1 only has one gap. There are several algorithms to calculate what is called the alignment score. Here, we refer to [58], [59] and [60], where the most common scoring systems are explained in detail.

Putting these two factors together, (the big amount of data generated by sequencers and the high computational cost required to carry out the alignment), help us to understand why the development of parallel tools for the alignment process is so important.

Most of state of the art aligners exploit parallelism on shared memory systems, so they have restrictions regarding the number of cores that can be used. Only some of them take advantage of HPC and Big Data solutions.

- **BWA** [61, 62, 63]: The state of the art tool par excellence regarding sequence alignment is the Burrows-Wheeler Aligner (BWA). It is included in various GATK best practices pipelines and it is widely accepted among the bioinformatics and genomics community. This tool includes thread-level parallelism in shared memory. It is developed in C with three different algorithms available.

- **Bowtie** [64]: Ultrafast, memory-efficient alignment program for aligning short DNA sequence reads to large genomes. It was developed at the University of Maryland in 2009. As BWA, includes thread level parallelism.

- **SOAP** [65, 66, 67, 68]: Program developed for efficient gapped and ungapped alignment of short oligonucleotides onto reference sequences. It includes GPU support since version 3.

- **Halvade** [69]: Hadoop-based aligner. It includes a variant detection phase which is the next stage after the sequence alignment in some DNA sequencing workflows.

- **SEAL** [70]: Pydoop [71], based aligner. It follows the MapReduce model, and allows users to write their programs in Python, calling BWA methods by means of a wrapper.

- **CUSHAW** [72, 73, 74]: CUDA compatible short read alignment algorithm for multiple GPUs sharing a single host. This aligner is designed based on the Burrows-Wheeler transform (BWT) and written using CUDA C++ parallel programming language.

- **pBWA** [75]: MPI implementation of BWA.

In this work, BWA has been used as the core of two alignment tools developed using Big Data technologies. The first one is BigBWA [76], which takes advantage of Hadoop as Big Data technology to increase the performance of BWA. The main advantages of our tool are the following. First, the alignment process is performed in parallel using a tested and scalable technology, which reduces the execution times dramatically. Second, BigBWA is fault tolerant, exploiting the fault tolerance capabilities of the underlying Big Data technology

on which it is based. And finally, no modifications to BWA are required to use BigBWA. As a consequence, any release of BWA (future or legacy) will be compatible with BigBWA.

Hadoop applications are typically developed in Java, but BWA is implemented in C. To overcome this issue BigBWA uses the Java Native Interface (JNI) [77], which allows the incorporation of native code written in programming languages such as C and C++, as well as code written in Java. Two independent software layers were created in BigBWA. The first one corresponds to the BWA software package, while the other is, strictly speaking, our tool. This design avoids any modification of the BWA source code, which assures the compatibility of BigBWA with any BWA version.

In order to validate our tool several experiments have been carried out on a cluster deployed in Amazon Web Services. Input data correspond to human DNA sequences from the *1000 Genomes* project. Results from these experiments can be observed in Chapter 3, here, let's illustrate the benefits of BigBWA with the following example. The Illumina HiSeqX^{TM} is able to generate 6 billion (6×10^9) reads. It means more than 40 days to perform the alignment phase onto a reference genome with the sequential version of BWA (1 thread). This time can be reduced to 5 days by using the multi-thread BWA version, and less than one day by running BigBWA on a small cluster. In the case of a medium-size cluster, when using BigBWA, this time is reduced to less than 5 hours, it is, 192$\times$ faster than BWA single-thread. At the same time, it is faster than other state of the art tools.

Regarding the second alignment tool developed in this thesis, we have developed the software SparkBWA [78]. SparkBWA follows the same philosophy than BigBWA, but the considered Big Data technology is Spark instead of Hadoop. SparkBWA was designed to meet three requirements. First, SparkBWA should outperform BWA and other BWA-based aligners both in terms of performance and scalability. The second requirement is related to keep the compatibility of SparkBWA with future and legacy versions of BWA. Since BWA is constantly evolving to include new functionalities and algorithms, it is important for SparkBWA to be agnostic regarding the BWA version. This is an important difference with respect to other existent tools based on BWA, which require modifications of the BWA source code. Finally, NGS professionals demand solutions to perform sequence alignments efficiently in such a way that the implementation details are completely hidden to them. For this reason SparkBWA provides a simple and flexible API to handle all the aspects related to the alignment process. In this way, bioinformaticians only need to focus on the scientific problem to deal with.

SparkBWA has been evaluated both in terms of performance, scalability and memory consumption, and a thorough comparison between SparkBWA and several state-of-art BWA-based aligners is also provided. Those tools take advantage of different parallel approaches as Pthreads, MPI, and Hadoop to improve the performance of BWA. Performance results demonstrate the benefits of our proposal. The evaluation shows that SparkBWA is almost twice faster than other state of the art tools. More precisely, it is 1.7× faster than SEAL, 1.4× faster than BigBWA or Halvade, and 1.25× faster than pBWA. SparkBWA reachs a speedup of 86× when using 128 cores regarding the BWA sequential version. Details about the design, implementation and performance results are shown in Chapter 4.

Multiple sequence alignment

The goal of protein sequence comparison is to discover structural or functional similarities among proteins. Biologically similar proteins may not exhibit a strong sequence similarity, but we would still like to recognize resemblance even when the sequences share only weak similarities. If the sequence similarity is weak, pairwise alignment can fail to identify biologically related sequences because weak pairwise similarities may fail statistical tests for significance. However, simultaneous comparison of many sequences often allows one to find similarities that are invisible in pairwise sequence comparison [60]. This is whats is called Multiple Sequence Alignment (MSA). MSA is an extension of the pairwise alignment to incorporate more than two sequences at a time. In many cases, the input set of query sequences are assumed to have an evolutionary relationship by which they share a lineage and are descended from a common ancestor. Multiple sequence alignments are computationally difficult to produce and most formulations of the problem lead to NP-complete combinatorial optimization problems. In this way, MSA is essential in order to predict the structure and function of proteins and RNAs, estimate phylogeny, and other common tasks in sequence analysis.

PASTA [79] is a tool, based on SATé [80], which produces highly accurate alignments, improving the accuracy and scalability of other state-of-art methods. PASTA is based on a workflow composed of several steps. During each phase, an external tool is called to perform different operations such as estimating an initial alignment and tree, computing MSAs on subsets of the original sequence set, or estimating the maximum likehood tree on a previously obtained MSA. Note that computing the MSAs is the most time consuming phase, implying in some cases over 70% of the total execution time.

PASTA is a multithreaded application that only supports shared memory computers. In

this way, PASTA is limited to process small or medium size input datasets, because the memory and time requirements of large datasets exceed the computing power of any shared memory system. In this thesis we introduce PASTASpark [81], an extension to PASTA that allows to execute it on a distributed memory cluster making use of Apache Spark.

We have used the Spark Python API (known as PySpark) to implement PASTASpark. The design of PASTASpark minimizes the changes in the original PASTA code. In fact, the same software can be used to run the unmodified PASTA on a multicore machine or PASTASpark on a cluster.

Regarding the performance results, it is important to remark here that only the most time consuming PASTA phase has been parallelized. The other phases are executed in the Spark driver. With this factor in mind, the speedup obtained in the CESGA Big Data cluster with 64 cores is 10.5× with respect to the single threaded PASTA. This number seems to be small at first sight, but, actually, it is very close to the upper limit predicted by the Amdahl's law. Our solution is able to process a 200K sequences dataset in less than 24 hours. Considering the original PASTA tool it was not possible to complete this process because of memory restrictions. More details about our approach can be found in Chapter 5.

Some of the state of the art tools to perform MSA using a parallel approach are presented below. To the best of our knowledge, there are no other tools that perform MSA by using Big Data technologies.

- **MAFFT** [82]: MSA program for amino acid or nucleotide sequences. The software is named after the acronym Multiple Alignment using Fast Fourier. Is written in C language and allows multiple threads.

- **MSAProbs** [83]: Tool to perform MSA for protein sequences. It is based on a combination of pair hidden Markov models and partition functions. It is developed in C++ and uses OpenMP to parallelize at thread level with a shared memory approach.

- **MSAProbs-MPI** [84]: MPI implementation of MSAProbs. It implements a distributed memory approach to MSAProbs by using MPI. In this way, it comes with a two-levels parallelism. One in distributed memory by using MPI, and other one in shared memory by using OpenMP.

- **QuickProbs** [85]: GPU based implementation of MSAProbs by using OpenCL.

1.5. Thesis outline

In the following chapters we provide the key articles that represent the main body of work for this thesis. In all these articles, the author of the thesis has been the main contributor. These articles have been either published in JCR journals or in high quality international conferences. The selection of publications has been made to delve into the main points mentioned in this introduction, and to have a more complete representation of the work carried out during this thesis. In Section 1.6, a full compendium of the journal and conference publications related to this thesis is presented.

In Chapter 2 we introduce Perldoop, a new tool developed in order to automatically translate Perl scripts into Hadoop ready Java codes. Our solution is very suitable for Natural Language Processing applications that are written in Perl code. In this way, Perl NLP codes can benefit from the parallel improvements that Hadoop provides. Results prove how using Perldoop the execution times are far better than using Hadoop Streaming with the original Perl codes.

In Chapter 3 we introduce BigBWA, a tool to align raw genomic sequences by using the state of the art software BWA. BigBWA uses Hadoop, HDFS and the MapReduce programming model in order to speed up the alignment process. By using BigBWA important improvements regarding the computation times are observed, while the correctness results compared with the original BWA are almost identical.

An evolution from BigBWA is SparkBWA, which is presented in Chapter 4. SparkBWA presents noticeable improvements in terms of performance and scalability regarding BigBWA and other state of the art tools. Also, it provides a simple and flexible API to handle all the aspects related to the alignment process from the Spark shell and uses Spark native functions to handle the input data.

We introduce PASTASpark in Chapter 5. PASTASpark is an efficient and parallel version of PASTA that uses Spark as Big Data engine. Note that only the alignment step runs into the Spark executors, which is the the most time consuming part in PASTA.

Finally, conclusions future work and some ideas of how the Big Data and the HPC world can converge are presented in Chapter 6.

1.6. List of publications

Next a list of publications derived from the work developed in this thesis is shown:

Articles in peer reviewed journals:

- P. Gamallo, J. C. Pichel, M. Garcia, J. M. Abuín, and T. F. Pena, "Análisis morfosintáctico y clasificación de entidades nombradas en un entorno Big Data," *Procesamiento del Lenguaje Natural*, vol. 53, pp. 17–24, 2014

- J. M. Abuín, J. C. Pichel, T. F. Pena, and J. Amigo, "BigBWA: Approaching the Burrows–Wheeler Aligner to Big Data Technologies," *Bioinformatics*, vol. 31, no. 24, pp. 4003–4005, 2015
  Impact factor (JCR 2015): 5.766. Decil 1 Q1.
  Category: MATHEMATICAL & COMPUTATIONAL BIOLOGY. Rank: **3/56**.

- J. M. Abuín, J. C. Pichel, T. F. Pena, and J. Amigo, "SparkBWA: speeding up the alignment of high-throughput DNA sequencing data," *PloS one*, vol. 11, no. 5, p. e0155461, 2016
  Impact factor (JCR 2016): 2.806. Q1.
  Category: MULTIDISCIPLINARY SCIENCES. Rank: **15/64**.

- J. M. Abuín, T. F. Pena, and J. C. Pichel, "PASTASpark: multiple sequence alignment meets Big Data," *Bioinformatics*
  Impact factor (JCR 2016): 7.307. Decil 1 Q1.
  Category: MATHEMATICAL & COMPUTATIONAL BIOLOGY. Rank: **2/57**.

Articles published in international conferences:

- J. M. Abuín, J. C. Pichel, T. F. Pena, P. Gamallo, and M. Garcia, "Perldoop: Efficient execution of Perl scripts on Hadoop clusters," in *IEEE International Conference on Big Data*, 2014, pp. 766–771

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

Chapter 2

Perldoop: Efficient Execution of Perl Scripts on Hadoop Clusters

J. M. Abuín, J. C. Pichel, T. F. Pena, P. Gamallo, and M. Garcia, "Perldoop: Efficient execution of Perl scripts on Hadoop clusters," in *IEEE International Conference on Big Data*, 2014, pp. 766–771

https://doi.org/10.1109/BigData.2014.7004303

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

CHAPTER 3

BIGBWA: APPROACHING THE BURROWS-WHEELER ALIGNER TO BIG DATA TECHNOLOGIES

J. M. Abuín, J. C. Pichel, T. F. Pena, and J. Amigo, "BigBWA: Approaching the Burrows–Wheeler Aligner to Big Data Technologies," *Bioinformatics*, vol. 31, no. 24, pp. 4003–4005, 2015

https://doi.org/10.1093/bioinformatics/btv506

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

Chapter 4

SparkBWA: Speeding Up the Alignment of High-Throughput DNA Sequencing Data

J. M. Abuín, J. C. Pichel, T. F. Pena, and J. Amigo, "SparkBWA: speeding up the alignment of high-throughput DNA sequencing data," *PloS one*, vol. 11, no. 5, p. e0155461, 2016

https://doi.org/10.1371/journal.pone.0155461

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

Chapter 5

PASTASpark: multiple sequence alignment meets Big Data

J. M. Abuín, T. F. Pena, and J. C. Pichel, "PASTASpark: multiple sequence alignment meets Big Data," *Bioinformatics*

https://doi.org/10.1093/bioinformatics/btx354

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

CHAPTER 6

CONCLUSIONS

In the recent years Big Data tools and ecosystems have become the standard when analyzing or processing huge datasets. We find the cause in the benefits of these technologies: fault tolerance, use of high level programming languages, or their similarity with High Performance Computing (HPC) regarding its parallel philosophy. Many experts in the HPC area agree that the Big Data and the HCP ecosystems should converge, or at least, share some approaches in order to enter into the Exascale era. In this way, HPC should incorporate some features from Big Data technologies such as the fault tolerance or the fast data distribution.

In this thesis, we use Big Data technologies to deal with some scientific problems that are computationally intensive regarding execution time (typical in HPC problems) and, at the same time, have a large input data size (typical in Big Data), with the objective of improving the execution time, scalability and efficiency. Conclusions derived from this work, and presented in this chapter, can clarify where the barrier between these two paradigms stands, or even prove whether this barrier exists at all. In this way, we want to contribute to solve the key question that in recent times has arisen among the HPC community could be solved: should Big Data be considered part of the High Performance Computing field?

Next, the main conclusions derived from this work are summarized:

- The field of NLP needs scalable and efficient tools in order to process the huge amount of information available in the Big Data era. However, the state of the art tools are usually written in languages that are not suitable neither for HPC nor Big Data technologies. This is the case, for example, of the existent modules in the Linguakit repository, written in Perl language. Although Hadoop provides the Hadoop Streaming tool

to deal with these cases, it has shown a poor performance. To overcome this issue, we have developed Perldoop (Chapter 2), which is a tool that automatically translates Perl scripts prepared to be executed using Hadoop Streaming into Hadoop-ready Java codes. However, the objective was not to develop a powerful tool that allows to automatically translate any existent Perl code to Java, but a simple and easy-to-use tool that takes as input Perl codes written for Hadoop Streaming, follows a reduced number of additional programming rules, and produces Hadoop-ready Java codes. By using this tool, NLP programs can benefit from the scalability, performance and fault tolerance properties of Big Data technologies. To facilitate this job, the tool uses a system based on tagging the source code and templates. Results show how, by using the MapReduce-ready Java codes generated by Perldoop, the NERC modules from Linguakit are executed $8\times$ faster than using the Hadoop Streaming tool with the same number of CPUs. For instance, the original NERC modules require about 19 days to process the whole Wikipedia in Spanish language. This result is improved by the Hadoop Streaming tool to less than 16 hours when using 64 cores. However, considering the Perldoop generated codes, the time is noticeably reduced to less than 2 hours using the same number of cores.

- In the last years, the available biological data in a digital format has experimented a big and fast growth. This process has been possible thanks to the next-generation sequencing (NGS) technologies. These technologies have facilitated the extraction of huge quantities of DNA sequences, which will further promote the future growth of biological databases. However, the process of giving meaning to all this information is overcoming the computing capacity of a CPU. Due to this fact, bioinformatics tools need to be efficient and scalable; that is, they need to deal with an ever growing amount of data. One of the most important challenges in Bioinformatics is the sequence alignment process. In this thesis, we deal with this problem introducing the BigBWA tool (Chapter 3). BigBWA uses Hadoop as a Big Data technology, while internally it uses the C functions from the state of the art software BWA to perform the alignment phase. In this way, two independent software layers were created in BigBWA. The first one corresponds to the BWA software package, while the other is, strictly speaking, our tool. This design implies that no modification of the BWA source code is needed, which assures the compatibility of BigBWA with any BWA version. Also, BigBWA is fault tolerant. It is worth noting that considering the 6 billion (6×10^9) reads that a Illumina HiSeqX^{TM} Ten is able to generate, BigBWA is capable or performing the alignment in

just 5 hours considering a medium size cluster. That is, 192× faster than single-thread BWA.

- The Big Data world evolves very quickly as Apache Spark illustrates. Since its birth, destined to overcome the Apache Hadoop limitations, it has experimented an enormous growth regarding functionalities and improvements. Taking advantage of this technology, we have developed SparkBWA (Chapter 4). SparkBWA follows the philosophy of BigBWA in terms of software design. SparkBWA fulfills three requirements. First, SparkBWA outperforms BWA and other BWA-based aligners both in terms of performance and scalability. Second, it keeps the compatibility with future and legacy versions of BWA. Since BWA is constantly evolving to include new functionalities and algorithms, it is important for SparkBWA to be agnostic regarding the BWA version. This is an important difference with respect to other existing tools based on BWA, which require modifications of the BWA source code. Finally, NGS professionals demand solutions to perform sequence alignments efficiently, in such a way that the implementation details are completely hidden to them. For this reason SparkBWA provides a simple and flexible API to handle all the aspects related to the alignment process, which allows bioinformaticians to focus only on the scientific problem to deal with. In terms of performance, we must highlight that SparkBWA is almost twice faster than other state of the art tools.

- Multiple Sequence Alignment (MSA) is an extension of the pairwise alignment to incorporate more than two sequences at a time. Multiple sequence alignments are computationally difficult to produce and most formulations of the problem lead to NP-complete combinatorial optimization problems. PASTA is a multithreaded application which produces highly accurate alignments, improving the accuracy and scalability of other state-of-art methods. PASTA is limited to process small or medium size input datasets, because the memory and time requirements of large datasets exceed the computing power of any shared memory system. In this thesis, we introduce PASTASpark (Chapter 5), an extension to PASTA that allows to execute it on a distributed memory cluster making use of Apache Spark. The design of PASTASpark minimizes the changes in the original PASTA code. In fact, the same software can be used to run the unmodified PASTA on a multicore machine or PASTASpark on a cluster. We are able to process a dataset composed of 200.000 sequences in less than 24 hours. Note that the original PASTA tool can not complete this process because of memory restrictions.

- We consider that a two level programming model can be a solution for mixing the Big Data and HPC worlds. A high level programming interface should be used when accessing the data for reading or writing (for example, by using HDFS) with languages such as Java, Scala or Python, more suited for these kind of tasks. This should be complemented with a low level programming interface when dealing with performance, memory consumption restrictions, or high performance libraries. Here, the programming languages should be the classic HPC languages, such as C, C++ or Fortran. In this way, advantages from the two worlds can be obtained, with a minimum performance penalty, as we have demonstrated in Chapter 3 and 4. We have demonstrated that the efficient communication of these two layers can be performed by using the Java Native Interface (JNI), or calling native methods from Python.

- When applications are not written in languages suited for Big Data nor HPC technologies, the best option is porting the codes to another language. However, this is a tedious and hard job. In this way, source to source compilers can be a great help, as the case shown in Chapter 2.

- Memory consumption could be a problem in Big Data applications. This is mainly caused by the way containers are implemented in YARN. A container is basically a process running as a Java Virtual Machine (JVM), with the corresponding memory overhead and memory management issues related to the Java Garbage Collector. A solution for this can be to use JNI as well, since native methods can reserve and free memory that the Java Garbage Collector is not aware of. This is a good approach, but at the same time, it requires that the programmer reserves and frees memory according to the program requirements.

6.1. Future work

We present here a list of future tasks that can be carried out in order to continue the work started in this thesis:

- Current Big Data schedulers do not allow to incorporate resources such as GPUs or accelerators such as Intel Xeon Phi, which are typical components of current supercomputing nodes. Add them as resources in YARN or Mesos should be a great step.

- Metagenomic sequencing studies are becoming increasingly popular, including the sequencing of human microbiomes and diverse environments. A fundamental computational problem in this context is read classification. For example, the assignment of each read to a taxonomic label. Due to the large number of reads produced by modern high-throughput sequencing technologies and the rapidly increasing number of available reference genomes, current software tools suffer from either long runtimes, large memory requirements and/or low accuracy. Because of this, Big Data technologies seem suitable for this problem. A possibility for future works is to enter into the metagenomics scientific field.

- Development of a new Perldoop version. This new version will incorporate the possibility of translate codes into other kind of Big Data technologies, and not only Apache Hadoop. Among the possible candidates we can highlight Apache Spark and Apache Storm.

- PASTA has another bottleneck that could be improved. This bottleneck is the starting tree construction, which is performed by the FastTree-2 tool and only scales up until 4 cores. If this construction phase could be improved, it would cause a great impact in the field of Multiple Sequence Alignment. In order to do so, the construction algorithm should be completely redesigned.

- BigBWA, SparkBWA and PASTASpark perform some writing/reading to/from disk, with the consequent lose of performance. These operations could be avoided by passing the input sequence data directly to the BWA and PASTA functions from the Big Data side of the application. In order to do that, a C library to communicate the Big Data side of the application with its C codes counterparts needs to be created.

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

Bibliography

[1] IBM, "Big Data at the Speed of Business," http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html, [Online; accessed July, 2014].

[2] Cern, "CERN Web Site," https://home.cern, [Online; accessed May, 2017].

[3] "Square Kilometre Array Home Page," http://skatelescope.org/, [Online; accessed June, 2017].

[4] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, ser. OSDI'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 10–10.

[5] ——, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[6] Hadoop, "Apache Hadoop," http://hadoop.apache.org, [Online; accessed May, 2017].

[7] TOP500, "TOP500 List," https://www.top500.org/, [Online; accessed May, 2017].

[8] Cesga, "Galicia Supercomputing Centre Web Site," http://www.cesga.es/, [Online; accessed May, 2017].

[9] E. exchange, "Processing power compared," http://pages.experts-exchange.com/processing-power-compared/, [Online; accessed May, 2017].

[10] D. W. Walker and J. J. Dongarra, "MPI: a standard message passing interface," *Supercomputer*, vol. 12, pp. 56–68, 1996.

[11] L. Dagum and R. Menon, "Openmp: an industry standard api for shared-memory programming," *IEEE computational science and engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[12] "OpenACC web page," https://www.openacc.org/, [Online; accessed Apr, 2017].

[13] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," in *Proc. of the 2Nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud)*, 2010, pp. 10–10.

[14] H. Asaadi, D. Khaldi, and B. M. Chapman, "A Comparative Survey of the HPC and Big Data Paradigms: Analysis and Experiments," in *2016 IEEE International Conference on Cluster Computing, CLUSTER 2016, Taipei, Taiwan, September 12-16, 2016*, 2016, pp. 423–432.

[15] D. A. Reed and J. Dongarra, "Exascale Computing and Big Data," *Commun. ACM*, vol. 58, no. 7, pp. 56–68, Jun. 2015.

[16] Lustre, "Lustre File System," http://lustre.org, [Online; accessed May, 2017].

[17] G. Staples, "TORQUE Resource Manager," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, ser. SC '06. New York, NY, USA: ACM, 2006.

[18] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 2003, pp. 44–60.

[19] "Nvidia CUDA Home Page," http://www.nvidia.com/object/cuda_home_new.html, [Online; accessed May, 2017].

[20] "OpenCL Home Page," https://www.khronos.org/opencl/, [Online; accessed May, 2017].

[21] "LAPACK web page," http://www.netlib.org/lapack/, [Online; accessed Apr, 2017].

[22] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, "Efficient Management of Parallelism in Object Oriented Numerical Software Libraries," in *Modern Software Tools in Scientific Computing*, E. Arge, A. M. Bruaset, and H. P. Langtangen, Eds. Birkhäuser Press, 1997, pp. 163–202.

[23] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, ser. MSST '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10.

[24] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache Hadoop YARN: Yet Another Resource Negotiator," in *Proc. of the 4th Annual Symposium on Cloud Computing (SOCC)*, 2013, pp. 5:1–5:16.

[25] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A Platform for Fine-grained Resource Sharing in the Data Center," in *Proc. of the 8th USENIX Conference on Networked Systems Design and Implementation*, 2011, pp. 295–308.

[26] T. White, *Hadoop: The Definitive Guide*, 3rd ed. O'Reilly Media, Inc., 2012.

[27] "Apache Pig Home Page," https://pig.apache.org/, [Online; accessed June, 2017].

[28] "Apache Hive home page," http://hive.apache.org/, [Online; accessed December, 2016].

[29] HBase, "HBase," https://hbase.apache.org/, [Online; accessed May, 2017].

[30] I. F. Haddad, "PVFS: A Parallel Virtual File System for Linux Clusters," *Linux J.*, vol. 2000, no. 80es, Nov. 2000.

[31] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '03. New York, NY, USA: ACM, 2003, pp. 29–43.

[32] M. K. McKusick and S. Quinlan, "GFS: Evolution on Fast-forward," *Queue*, vol. 7, no. 7, pp. 10:10–10:20, Aug. 2009.

[33] T. White, *Hadoop: The Definitive Guide*, 4th ed. O'Reilly Media, Inc., 2015.

[34] S. N. Srirama, P. Jakovits, and E. Vainikko, "Adapting Scientific Computing Problems to Clouds Using MapReduce."

[35] M. Islam, A. K. Huang, M. Battisha, M. Chiang, S. Srinivasan, C. Peters, A. Neumann, and A. Abdelnur, "Oozie: towards a scalable workflow management system for Hadoop," in *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*. ACM, 2012, p. 4.

[36] "Cascading home page," http://www.cascading.org/, [Online; accessed February, 2016].

[37] M. Ding, L. Zheng, Y. Lu, L. Li, S. Guo, and M. Guo, "More convenient more overhead: the performance evaluation of Hadoop streaming," in *ACM Symp. on Research in Applied Computation*, 2011, pp. 307–313.

[38] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, *Learning Spark: Lightning-Fast Big Data Analytics*, 1st ed. O'Reilly Media, Inc., 2015.

[39] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing," in *Proc. of the 9th USENIX Conference on Networked Systems Design and Implementation*, 2012, pp. 2–2.

[40] R. Agerri, X. Artola, Z. Beloki, G. Rigau, and A. Soroa, "Big data for natural language processing: a streaming approach," *Knowledge-Based Systems*, vol. 79, pp. 36–42, 2015.

[41] P. Gamallo and M. García, "A Resource-Based Method for Named Entity Extraction and Classification," *LNCS series*, vol. 7026, pp. 610–623, 2011.

[42] T. Hasegawa, S. Sekine, and R. Grishman, "Discovering relations among named entities from large corpora," in *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 2004, p. 415.

[43] Z. Kozareva, J. Silva, P. Gamallo, and G. Lopes, "Cluster analysis of named entities," in *Intelligent Information Processing and Web Mining*. Springer, 2004, pp. 429–433.

[44] J. Nothman, N. Ringland, W. Radford, T. Murphy, and J. R. Curran, "Learning multilingual named entity recognition from Wikipedia," *Artificial Intelligence*, vol. 194, pp. 151–175, 2013.

[45] D. Lin and X. Wu, "Phrase clustering for discriminative learning," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International*

Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2. Association for Computational Linguistics, 2009, pp. 1030–1038.

[46] J. R. Finkel, T. Grenager, and C. Manning, "Incorporating non-local information into information extraction systems by gibbs sampling," in *Proceedings of the 43rd annual meeting on association for computational linguistics*. Association for Computational Linguistics, 2005, pp. 363–370.

[47] M. Garcia and P. Gamallo, "An Entity-Centric Coreference Resolution System for Person Entities with Rich Linguistic Information," in *COLING*, 2014, pp. 741–752.

[48] ——, "Yet another suite of multilingual NLP tools," in *International Symposium on Languages, Applications and Technologies*. Springer, 2015, pp. 65–75.

[49] J. M. Abuín, J. C. Pichel, T. F. Pena, P. Gamallo, and M. Garcia, "Perldoop: Efficient execution of Perl scripts on Hadoop clusters," in *IEEE International Conference on Big Data*, 2014, pp. 766–771.

[50] D. A. Benson, M. Cavanaugh, K. Clark, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and E. W. Sayers, "GenBank," *Nucleic Acids Research*, vol. 41, no. D1, p. D36, 2013.

[51] R. Apweiler, A. Bairoch, C. H. Wu, W. C. Barker, B. Boeckmann, S. Ferro, E. Gasteiger, H. Huang, R. Lopez, M. Magrane, M. J. Martin, D. A. Natale, C. O'Donovan, N. Redaschi, and L. L. Yeh, "UniProt: the Universal Protein knowledgebase," *Nucleic Acids Research*, vol. 32, no. S1, pp. 115–119, 2004.

[52] R. Li, H. Zhu, J. Ruan, W. Qian, X. Fang, Z. Shi, Y. Li, S. Li, G. Shan, K. Kristiansen *et al.*, "De novo assembly of human genomes with massively parallel short read sequencing," *Genome research*, vol. 20, no. 2, pp. 265–272, 2010.

[53] J. Shendure, R. D. Mitra, C. Varma, and G. M. Church, "Advanced sequencing technologies: methods and goals," *Nature Reviews Genetics*, vol. 5, no. 5, pp. 335–344, 2004.

[54] B. Schmidt, *Bioinformatics: High Performance Parallel Computer Architectures*, 1st ed. CRC Press, 2011.

[55] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly, and M. A. DePristo, "The Genome

Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data," *Genome Research*, vol. 20, pp. 1297–1303, Sep 2010.

[56] B. institute, "GATK Best Practices," https://software.broadinstitute.org/gatk/best-practices/, [Online; accessed May, 2017].

[57] D. W. Mount, *Bioinformatics: sequence and genome analysis*. CSHL press, 2004, ch. Phylogenetic Prediction.

[58] R. C. Deonier, S. Tavaré, and M. Waterman, *Computational genome analysis: an introduction*. Springer Science & Business Media, 2005.

[59] B. Haubold and T. Wiehe, *Introduction to computational biology: an evolutionary approach*. Springer Science & Business Media, 2006.

[60] N. C. Jones and P. Pevzner, *An introduction to bioinformatics algorithms*. MIT press, 2004.

[61] H. Li and R. Durbin, "Fast and Accurate Short Read Alignment with Burrows-Wheeler Transform," *Bioinformatics*, vol. 25, no. 14, pp. 1754–1760, 2009.

[62] ——, "Fast and Accurate Long-Read Alignment with Burrows-Wheeler Transform," *Bioinformatics*, vol. 26, no. 5, pp. 589–595, 2010.

[63] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM," *arXiv:1303.3997v2*, 2013.

[64] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome biology*, vol. 10, no. 3, p. R25, 2009.

[65] R. Li, Y. Li, K. Kristiansen, and J. Wang, "SOAP: short oligonucleotide alignment program," *Bioinformatics*, vol. 24, no. 5, p. 713, 2008.

[66] R. Li, C. Yu, Y. Li, T.-W. Lam, S.-M. Yiu, K. Kristiansen, and J. Wang, "SOAP2: an improved ultrafast tool for short read alignment," *Bioinformatics*, vol. 25, no. 15, p. 1966, 2009.

[67] C.-M. Liu, T. K. F. Wong, E. Wu, R. Luo, S.-M. Yiu, Y. Li, B. Wang, C. Yu, X. Chu, K. Zhao, R. Li, and T. W. Lam, "SOAP3: ultra-fast GPU-based parallel alignment tool for short reads," *Bioinformatics*, vol. 28, no. 6, pp. 878–879, 2012.

[68] R. Luo, T. Wong, J. Zhu, C.-M. Liu, X. Zhu, E. Wu, L.-K. Lee, H. Lin, W. Zhu, D. W. Cheung, H.-F. Ting, S.-M. Yiu, S. Peng, C. Yu, Y. Li, R. Li, and T.-W. Lam, "SOAP3-dp: Fast, Accurate and Sensitive GPU-Based Short Read Aligner," *PLoS ONE*, vol. 8, no. 5, 2013.

[69] D. Decap, J. Reumers, C. Herzeel, P. Costanza, and J. Fostier, "Halvade: Scalable Sequence Analysis with MapReduce," *Bioinformatics*, vol. 31, no. 15, pp. 2482–2488, 2015.

[70] L. Pireddu, S. Leo, and G. Zanetti, "SEAL: a distributed short read mapping and duplicate removal tool," *Bioinformatics*, vol. 27, no. 15, pp. 2159–2160, 2011.

[71] S. Leo and G. Zanetti, "Pydoop: a Python MapReduce and HDFS API for Hadoop," in *Proc. of 19th Symposyum on HPDC*, 2010, pp. 819–825.

[72] Y. Liu, B. Schmidt, and D. L. Maskell, "CUSHAW: a CUDA compatible short read aligner to large genomes based on the Burrows-Wheeler transform," *Bioinformatics*, vol. 28, no. 14, pp. 1830–1837, 2012.

[73] Y. Liu and B. Schmidt, "CUSHAW2-GPU: empowering faster gapped short-read alignment using GPU computing," *IEEE Design & Test*, vol. 31, no. 1, pp. 31–39, 2014.

[74] Y. Liu, B. Popp, and B. Schmidt, "CUSHAW3: sensitive and accurate base-space and color-space short-read alignment with hybrid seeding," *PloS one*, vol. 9, no. 1, p. e86869, 2014.

[75] D. Peters, X. Luo, K. Qiu, and P. Liang, "Speeding Up Large-Scale Next Generation Sequencing Data Analysis with pBWA," *Journal of Applied Bioinformatics & Computational Biology*, vol. 1, no. 1, pp. 1–6, 2012.

[76] J. M. Abuín, J. C. Pichel, T. F. Pena, and J. Amigo, "BigBWA: Approaching the Burrows–Wheeler Aligner to Big Data Technologies," *Bioinformatics*, vol. 31, no. 24, pp. 4003–4005, 2015.

[77] S. Liang, *Java Native Interface: Programmer's Guide and Reference*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[78] J. M. Abuín, J. C. Pichel, T. F. Pena, and J. Amigo, "SparkBWA: speeding up the alignment of high-throughput DNA sequencing data," *PloS one*, vol. 11, no. 5, p. e0155461, 2016.

[79] Mirarab, S. *et al.*, "PASTA: ultra-large multiple sequence alignment for nucleotide and amino-acid sequences," *Journal of Computational Biology*, vol. 22, no. 5, pp. 377–386, 2015.

[80] Liu, K. *et al.*, "SATé-II: very fast and accurate simultaneous estimation of multiple sequence alignments and phylogenetic trees," *Systematic Biology*, vol. 61, no. 1, pp. 90–106, 2012.

[81] J. M. Abuín, T. F. Pena, and J. C. Pichel, "PASTASpark: multiple sequence alignment meets Big Data," *Bioinformatics*.

[82] Katoh, K. *et al.*, "MAFFT: improvement in accuracy of multiple sequence alignment," *Nucleic acids research*, vol. 33, no. 2, pp. 511–518, 2005.

[83] Y. Liu, B. Schmidt, and D. L. Maskell, "MSAProbs: multiple sequence alignment based on pair hidden Markov models and partition function posterior probabilities," *Bioinformatics*, vol. 26, no. 16, pp. 1958–1964, 2010.

[84] J. González-Domínguez, Y. Liu, J. Touriño, and B. Schmidt, "MSAProbs-MPI: parallel multiple sequence aligner for distributed-memory systems," *Bioinformatics*, vol. 32, no. 24, pp. 3826–3828, 2016.

[85] A. Gudyś and S. Deorowicz, "QuickProbs—A Fast Multiple Sequence Alignment Algorithm Designed for Graphics Processors," *PLOS ONE*, vol. 9, no. 2, pp. 1–18, 02 2014.

[86] P. Gamallo, J. C. Pichel, M. Garcia, J. M. Abuín, and T. F. Pena, "Análisis morfosintáctico y clasificación de entidades nombradas en un entorno Big Data," *Procesamiento del Lenguaje Natural*, vol. 53, pp. 17–24, 2014.

List of Figures

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

List of Tables